# Partition Tree Weighting

Joel Veness[‡†] Martha White[†] Michael Bowling[†] András György[†]

[†]University of Alberta, Edmonton, Canada
[‡]DeepMind Technologies

**Abstract**

This paper introduces the Partition Tree Weighting technique, an efficient meta-algorithm for piecewise stationary sources. The technique works by performing Bayesian model averaging over a large class of possible partitions of the data into locally stationary segments. It uses a prior, closely related to the Context Tree Weighting technique of Willems, that is well suited to data compression applications. Our technique can be applied to any coding distribution at an additional time and space cost only logarithmic in the sequence length. We provide a competitive analysis of the redundancy of our method, and explore its application in a variety of settings. The order of the redundancy and the complexity of our algorithm matches those of the best competitors available in the literature, and the new algorithm exhibits a superior complexity-performance trade-off in our experiments.

## 1 Introduction

Coping with data generated from non-stationary sources is a fundamental problem in data compression. Many real-world data sources drift or change suddenly, often violating the stationarity assumptions implicit in many models. Rather than modifying such models so that they robustly handle non-stationary data, one promising approach has been to instead design meta-algorithms that automatically generalize existing stationary models to various kinds of non-stationary settings.

A particularly well-studied kind of non-stationary source is the class of piecewise stationary sources. Algorithms designed for this setting assume that the data generating source can be well modeled by a sequence of stationary sources. This assumption is quite reasonable, as piecewise stationary sources have been shown [1] to adequately handle various types of non-stationarity. Piecewise stationary sources have received considerable attention from researchers in information theory [21, 20, 16], online learning [11, 19, 12, 6, 10, 9], time series [1, 5], and graphical models [7, 15, 2].

An influential approach for piecewise stationary sources is the universal *transition diagram* technique of Willems [21] for the statistical data compression setting. This technique performs Bayesian model averaging over all possible partitions of a sequence of data. Though powerful for modeling piecewise stationary sources, its quadratic time complexity makes it too computationally intensive for many applications. Since then, more efficient algorithms have been introduced that weight over restricted subclasses of partitions [20, 16, 10, 9]. For example, Live and Die Coding [20] considers only $\log t$ partitions at any particular time $t$ by terminating selected partitions as

time progresses, resulting in an $O(n \log n)$ algorithm for binary, piecewise stationary, memoryless sources with provable redundancy guarantees.[1] György et al. [9] recently extended and generalized these approaches into a parametrized online learning framework that can interpolate between many of the aforementioned weighting schemes for various loss functions. We also note that linear complexity algorithms exist for prediction in changing environments [11, 12, 23], but these algorithms work with a restricted class of predictors and are not applicable directly to the data compression problem considered here.

In this paper we introduce the Partition Tree Weighting (PTW) technique, a computationally efficient meta-algorithm that also works by weighting over a large subset of possible partitions of the data into stationary segments. Compared with previous work, the distinguishing feature of our approach is to use a prior, closely related to Context Tree Weighting [22], that contains a strong bias towards partitions containing long runs of stationary data. As we shall see later, this bias is particularly suited for data compression applications, while still allowing us to provide theoretical guarantees competitive with previous low complexity weighting approaches.

## 2 Background

We begin with some terminology for sequential, probabilistic data generating sources. An alphabet is a finite, non-empty set of symbols, which we will denote by $\mathcal{X}$. A string $x_1 x_2 \dots x_n \in \mathcal{X}^n$ of length $n$ is denoted by $x_{1:n}$. The prefix $x_{1:j}$ of $x_{1:n}$, $j \leq n$, is denoted by $x_{\leq j}$ or $x_{<j+1}$. The empty string is denoted by $\epsilon$. Our notation also generalises to out of bounds indices; that is, given a string $x_{1:n}$ and an integer $m > n$, we define $x_{1:m} := x_{1:n}$ and $x_{m:n} := \epsilon$. The concatenation of two strings $s$ and $r$ is denoted by $sr$.

**Probabilistic Data Generating Sources.** A probabilistic data generating source $\rho$ is defined by a sequence of probability mass functions $\rho_n : \mathcal{X}^n \to [0, 1]$, for all $n \in \mathbb{N}$, satisfying the compatibility constraint that $\rho_n(x_{1:n}) = \sum_{y \in \mathcal{X}} \rho_{n+1}(x_{1:n} y)$ for all $x_{1:n} \in \mathcal{X}^n$, with base case $\rho_0(\epsilon) = 1$. From here onwards, whenever the meaning is clear from the argument to $\rho$, the subscripts on $\rho$ will be dropped. Under this definition, the conditional probability of a symbol $x_n$ given previous data $x_{<n}$ is defined as $\rho(x_n | x_{<n}) := \rho(x_{1:n}) / \rho(x_{<n})$ provided $\rho(x_{<n}) > 0$, with the familiar chain rules $\rho(x_{1:n}) = \prod_{i=1}^{n} \rho(x_i | x_{<i})$ and $\rho(x_{i:j} | x_{<i}) = \prod_{k=i}^{j} \rho(x_k | x_{<k})$ now following.

**Temporal Partitions.** Now we introduce some notation to describe temporal partitions. A segment is a tuple $(a, b) \in \mathbb{N} \times \mathbb{N}$ with $a \leq b$. A segment $(a, b)$ is said to overlap with another segment $(c, d)$ if there exists an $i \in \mathbb{N}$ such that $a \leq i \leq b$ and $c \leq i \leq d$. A temporal partition $\mathcal{P}$ of a set of time indices $S = \{1, 2, \dots n\}$, for some $n \in \mathbb{N}$, is a set of non-overlapping segments such that for all $x \in \mathcal{S}$, there exists a segment $(a, b) \in \mathcal{P}$ such that $a \leq x \leq b$. We also use the overloaded notation $\mathcal{P}(a, b) := \{(c, d) \in \mathcal{P} \ : \ a \leq c \leq d \leq b\}$. Finally, $\mathcal{T}_n$ will be used to denote the set of all possible temporal partitions of $\{1, 2, \dots, n\}$.

**Piecewise Stationary Sources.** We can now define a piecewise stationary data generating source $\mu$ in terms of a partition $\mathcal{P} = \{(a_1, b_1), (a_2, b_2), \dots\}$ and a set of probabilistic data generating sources $\{\mu^1, \mu^2, \dots\}$, such that for all $n \in \mathbb{N}$, for all $x_{1:n} \in \mathcal{X}^n$,

$$\mu(x_{1:n}) := \prod_{(a,b) \in \mathcal{P}_n} \mu^{f(a)}(x_{a:b}),$$

[1] All logarithms in this paper are of base 2.

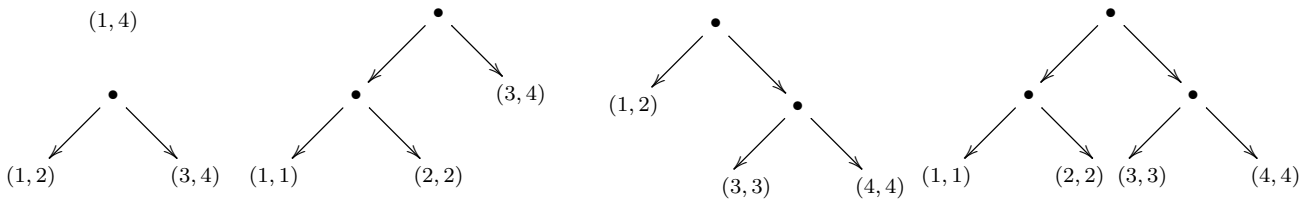



Figure 1: The set $\mathcal{C}_2$ represented as a collection of partition trees.

where $\mathcal{P}_n := \{(a_i, b_i) \in \mathcal{P} \,:\, a_i \leq n\}$ and $f(i)$ returns the index of the time segment containing $i$, that is, it gives a value $k \in \mathbb{N}$ such that both $(a_k, b_k) \in \mathcal{P}$ and $a_k \leq i \leq b_k$.

**Redundancy.** The ideal code length given by a probabilistic model (or probability assignment) $\rho$ on a data sequence $x_{1:n} \in \mathcal{X}^n$ is given by $-\log \rho(x_{1:n})$, and the redundancy of $\rho$, with respect to a probabilistic data generating source $\mu$, is defined as $\log \mu(x_{1:n}) - \log \rho(x_{1:n})$. This quantity corresponds to the amount of extra bits we would need to transmit $x_{1:n}$ using an optimal code designed for $\rho$ (assuming the ideal code length) compared to using an optimal code designed for the data generating source $\mu$.

## 3 Partition Tree Weighting

Almost all existing prediction algorithms designed for changing source statistics are based on the transition diagram technique of Willems [21]. This technique performs exact Bayesian model averaging over the set of temporal partitions, or more precisely, the method averages over all coding distributions formed by employing a particular base model $\rho$ on all segments of every possible partition. Averaging over all temporal partitions (also known as transition paths) results in an algorithm of $O(n^2)$ complexity. Several reduced complexity methods were proposed in the literature that average over a significantly smaller set of temporal partitions [20, 10, 9]: the reduced number of partitions allows for the computational complexity to be pushed down to $O(n \log n)$, while still being sufficiently rich to guarantee almost optimal redundancy behavior (typically $O(\log n)$ times larger than the optimum, for lossless data compression). In this paper we propose another member of this family of methods. Our reduced set of temporal partitions, as well as the corresponding mixture weights, are obtained from the Context Tree Weighting (CTW) algorithm [22], which results in similar theoretical guarantees as the other methods, but shows superior performance in all of our experiments. The method, called Partition Tree Weighting, heavily utilizes the computational advantages offered by the CTW algorithm.

We now derive the Partition Tree Weighting (PTW) technique. As PTW is a meta-algorithm, it takes as input a base model which we denote by $\rho$ from here onwards. This base model determines what kind of data generating sources can be processed.

### 3.1 Model Class

We begin by defining the class of binary temporal partitions. Although more restrictive than the class of all possible temporal partitions, binary temporal partitions possess important computational advantages that we will later exploit.

**Definition 1.** *Given a depth parameter $d \in \mathbb{N}$ and a time $t \in \mathbb{N}$, the set $\mathcal{C}_d(t)$ of all binary temporal partitions from $t$ is recursively defined by*

$$\mathcal{C}_d(t) := \left\{\{(t, t + 2^d - 1)\}\right\} \cup \left\{\mathcal{S}_1 \cup \mathcal{S}_2 : \mathcal{S}_1 \in \mathcal{C}_{d-1}\left(t\right), \mathcal{S}_2 \in \mathcal{C}_{d-1}\left(t + 2^{d-1}\right)\right\},$$

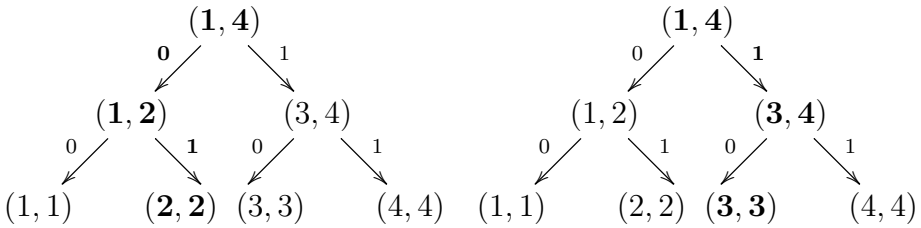



Figure 2: Partitions updated at $t = 2$ (left) and $t = 3$ (right) in a depth-2 partition tree.

*with* $\mathcal{C}_0(t) := \{\{(t, t)\}\}$. *Furthermore, we define* $\mathcal{C}_d := \mathcal{C}_d(1)$.

For example, $\mathcal{C}_2 = \{\ \{(1,4)\},\ \{(1,2),(3,4)\},\ \{(1,1),(2,2),(3,4)\},\ \{(1,2),(3,3),(4,4)\},\ \{(1,1),(2,2),(3,3),(4,4)\}\ \}$. Each partition can be naturally mapped onto a tree structure which we will call a partition tree. Figure 1 shows the collection of partition trees represented by $\mathcal{C}_2$. Notice that the number of binary temporal partitions $|\mathcal{C}_d|$ grows roughly double exponentially in $d$. For example, $|\mathcal{C}_0| = 1$, $|\mathcal{C}_1| = 2$, $|\mathcal{C}_2| = 5$, $|\mathcal{C}_3| = 26$, $|\mathcal{C}_4| = 677$, $|\mathcal{C}_5| = 458330$, which means that some ingenuity will be required to weight over all $\mathcal{C}_d$ efficiently.

### 3.2 Coding Distribution

We now consider a particular weighting over $\mathcal{C}_d$ that has both a bias towards simple partitions and efficient computational properties. Given a data sequence $x_{1:n}$, we define

$$\text{PTW}_d(x_{1:n}) := \sum_{\mathcal{P} \in \mathcal{C}_d} 2^{-\Gamma_d(\mathcal{P})} \prod_{(a,b) \in \mathcal{P}} \rho(x_{a:b}), \tag{1}$$

where $\Gamma_d(\mathcal{P})$ gives the number of nodes in the partition tree associated with $\mathcal{P}$ that have a depth less than $d$. This prior weighting is identical to how the Context Tree Weighting method [22] weights over tree structures, and is an application of the general technique used by the class of Tree Experts described in Section 5.3 of [4]. It is a valid prior, as one can show $\sum_{\mathcal{P} \in \mathcal{C}_d} 2^{-\Gamma_d(\mathcal{P})} = 1$ for all $d \in \mathbb{N}$. Note that Algorithm 1 is a special case, using a prior $2^{-\Gamma_d(\mathcal{P})}$ for $\mathcal{P} \in \mathcal{C}_d$ and 0 otherwise, of the class of general algorithms discussed in [9]. As such, the main contribution of this paper is the introduction of this specific prior. A direct computation of Equation 1 is clearly intractable. Instead, an efficient approach can be obtained by noting that Equation 1 can be recursively decomposed.

**Lemma 1.** *For any depth* $d \in \mathbb{N}$*, given a sequence of data* $x_{1:n} \in \mathcal{X}^n$ *satisfying* $n \leq 2^d$*,*

$$\text{PTW}_d(x_{1:n}) = \frac{1}{2}\rho(x_{1:n}) + \frac{1}{2}\text{PTW}_{d-1}\left(x_{1:k}\right)\text{PTW}_{d-1}\left(x_{k+1:n}\right), \tag{2}$$

*where* $k = 2^{d-1}$.

*Proof.* A straightforward adaptation of [22, Lemma 2], see Appendix A. □

### 3.3 Algorithm

Lemma 1 allows us to compute $\text{PTW}_d(x_{1:n})$ in a bottom up fashion. This leads to an algorithm that runs in $O(nd)$ time *and* space by maintaining a context tree data structure in memory. One of our main contributions is to further reduce the space overhead of PTW to $O(d)$ by exploiting the regular access pattern to this data structure. To give some intuition, Figure 2 shows in bold the nodes in a context tree data structure that would need to be updated at times $t = 2$ and $t = 3$. The

**Algorithm 1** PARTITION TREE WEIGHTING - $\text{PTW}_d(x_{1:n})$

**Require:** A depth parameter $d \in \mathbb{N}$
**Require:** A data sequence $x_{1:n} \in \mathcal{X}^n$ satisfying $n \leq 2^d$
**Require:** A base probabilistic model $\rho$

$b_j \leftarrow 1, w_j \leftarrow 1, r_j \leftarrow 1,$ for $0 \leq j \leq d$
**for** $t = 1$ to $n$ **do**
  $i \leftarrow \text{MSCB}_d(t)$
  $b_i \leftarrow w_{i+1}$
  **for** $j = i + 1$ to $d$ **do**
    $r_j \leftarrow t$
  **end for**
  $w_d \leftarrow \rho(x_{r_d:t})$
  **for** $i = d - 1$ to $0$ **do**
    $w_i \leftarrow \frac{1}{2}\rho(x_{r_i:t}) + \frac{1}{2}w_{i+1}b_i$
  **end for**
**end for**
**return** $w_0$

key observation is that because our access patterns are performing a kind of depth first traversal of the context tree, the needed statistics can be summarized in a stack of size $d$. This has important practical significance, as the performance of many interesting base models will depend on how much memory is available.

Algorithm 1 describes our $O(nd)$ time and $O(d)$ space technique for computing $\text{PTW}_d(x_{1:n})$. It uses a routine, $\text{MSCB}_d(t)$, that returns the most significant changed context bit; that is, for $t > 1$, this is the number of bits to the left of the most significant location at which the $d$-bit binary representations of $t - 1$ and $t - 2$ differ, with $\text{MSCB}_d(1) := 0$ for all $d \in \mathbb{N}$. For example, for $d = 5$, we have $\text{MSCB}_5(4) = 4$ and $\text{MSCB}_5(7) = 3$. Since $d = \lceil \log n \rceil$, the algorithm effectively runs in $O(n \log n)$ time and $O(\log n)$ space. Furthermore, Algorithm 1 can be modified to run incrementally, as $\text{PTW}_d(x_{1:n})$ can be computed from $\text{PTW}_d(x_{<n})$ in $O(d)$ time provided the intermediate buffers $b_i$, $w_i$, $r_i$ for $0 \leq i \leq d$ are kept in memory.

### 3.4 Theoretical Properties

We now provide a theoretical analysis of the Partition Tree Weighting method. Our first result shows that using PTW with a base model $\rho$ is almost as good as using $\rho$ with any partition in the class $\mathcal{C}_d$.

**Proposition 1.** *For all $n \in \mathbb{N}$, where $d = \lceil \log n \rceil$, for all $x_{1:n} \in \mathcal{X}$, for all $\mathcal{P} \in \mathcal{C}_d$, we have*

$$-\log \text{PTW}_d(x_{1:n}) \leq \Gamma_d(\mathcal{P}) + \sum_{(a,b)\in\mathcal{P}} -\log \rho(x_{a:b}).$$

*Proof.* We have that

$$-\log \text{PTW}_d(x_{1:n}) = -\log \left( \sum_{\mathcal{P}\in\mathcal{C}_d} 2^{-\Gamma_d(\mathcal{P})} \prod_{(a,b)\in\mathcal{P}} \rho(x_{a:b}) \right) \leq \Gamma_d(\mathcal{P}) - \log \prod_{(a,b)\in\mathcal{P}} \rho(x_{a:b}),$$

where $\mathcal{P}$ is an arbitrary partition in $\mathcal{C}_d$. Rewriting the last term completes the proof. $\square$

The next result shows that there always exists a binary temporal partition (i.e., a partition in $\mathcal{C}_d$) which is in some sense close to any particular temporal partition. To make this more precise, we introduce some more terminology. First we define $C(\mathcal{P}) := \{a\}_{(a,b)\in\mathcal{P}} \setminus \{1\}$, which is the set of time indices where an existing segment ends and a new segment begins in partition $\mathcal{P}$. Now, if $C(\mathcal{P}) \subseteq C(\mathcal{P}')$, we say $\mathcal{P}'$ is a refinement of partition $\mathcal{P}$. In other words, $\mathcal{P}'$ is a refinement of $\mathcal{P}$ if $\mathcal{P}'$ always starts a new segment whenever $\mathcal{P}$ does. With a slight abuse of notation, we will also use $C(\mathcal{P})$ to denote the partition $\mathcal{P}$.

**Lemma 2.** *For all $n \in \mathbb{N}$, for any temporal partition $\mathcal{P} \in \mathcal{T}_n$, with $d = \lceil \log n \rceil$, there exists a binary temporal partition $\mathcal{P}' \in \mathcal{C}_d$ such that $\mathcal{P}'$ is a refinement of $\mathcal{P}$ and $|\mathcal{P}'| \leq |\mathcal{P}|(\lceil \log n \rceil + 1)$.*

*Proof.* We prove this via construction. Consider a binary tree $T_i$ with $1 \leq i \leq n$ formed from the following recursive procedure: 1. Set $a = 1$, $b = 2^d$, add the node $(a, b)$ to the tree. 2. If $a = i$ then stop; otherwise, add $(a, \lfloor \frac{b-a}{2} \rfloor)$, $(a + \lfloor \frac{b-a}{2} \rfloor + 1, b)$ as children to node $(a, b)$, and then set $(a, b)$ to the newly added child containing $i$ and goto step 2.

Next define $\mathcal{L}(T_i)$ and $\mathcal{I}(T_i)$ to be the set of leaf and internal nodes of $T_i$ respectively. Notice that $|\mathcal{L}(T_i)| \leq d + 1$ and that $\mathcal{L}(T_i) \in \mathcal{C}_d$. Now, consider the set

$$\mathcal{P}' := \left\{ (a, b) \in \bigcup_{i\in C(\mathcal{P})} \mathcal{L}(T_i) \,:\, (a, b) \notin \bigcup_{i\in C(\mathcal{P})} \mathcal{I}(T_i) \right\}.$$

It is easy to verify that $\mathcal{P}' \in \mathcal{C}_d$ and that $C(\mathcal{P}) \subseteq C(\mathcal{P}')$. The proof is concluded by noticing that

$$|\mathcal{P}'| \leq \left| \bigcup_{i\in C(\mathcal{P})} \mathcal{L}(T_i) \right| \leq \sum_{i\in C(\mathcal{P})} |\mathcal{L}(T_i)| \leq |C(\mathcal{P})|(d + 1) \leq |\mathcal{P}| \left(\lceil \log n \rceil + 1\right). \qquad \square$$

Next we show that if we have a redundancy bound for the base model $\rho$ that holds for any finite sequence of data generated by some class of bounded memory data generating sources, we can automatically derive a redundancy bound when using PTW on the piecewise stationary extension of that same class.

**Theorem 1.** *For all $n \in \mathbb{N}$, using* PTW *with $d = \lceil \log n \rceil$ and a base model $\rho$ whose redundancy is upper bounded by a non-negative, monotonically non-decreasing, concave function $g : \mathbb{N} \to \mathbb{R}$ with $g(0) = 0$ on some class $\mathcal{G}$ of bounded memory data generating sources, the redundancy*

$$\log \mu(x_{1:n}) - \log \text{PTW}_d(x_{1:n}) \leq \Gamma_d(\mathcal{P}') + |\mathcal{P}|\, g\left( \left\lceil \frac{n}{|\mathcal{P}|(\lceil \log n \rceil + 1)} \right\rceil \right) (\lceil \log n \rceil + 1),$$

*where $\mu$ is a piecewise stationary data generating source, and the data in each of the stationary regions $\mathcal{P} \in \mathcal{T}_n$ is distributed according to some source in $\mathcal{G}$. Furthermore, $\Gamma_d(\mathcal{P}')$ can be upper bounded independently of $d$ by $2|\mathcal{P}|\,(\lceil \log n \rceil + 1)$.*

*Proof.* From Lemma 2, we know that there exists a partition $\mathcal{P}' \in \mathcal{C}_d$ that is a refinement of $\mathcal{P}$ containing at most $|\mathcal{P}|(\lceil \log n \rceil + 1)$ segments. Applying Proposition 1 to $\mathcal{P}'$ and using our

redundancy bound $g$ gives

$$\begin{aligned}
\log \mu(x_{1:n}) - \log \mathrm{PTW}_d(x_{1:n}) &\leq \log \mu(x_{1:n}) + \Gamma_d(\mathcal{P}') - \sum_{(a,b)\in\mathcal{P}'} \log \rho(x_{a:b}) \\
&= \Gamma_d(\mathcal{P}') + \sum_{(a,b)\in\mathcal{P}} \log \mu^{f(a)}(x_{a:b}) - \sum_{(a,b)\in\mathcal{P}'} \log \rho(x_{a:b}) \\
&= \Gamma_d(\mathcal{P}') - \sum_{(a,b)\in\mathcal{P}'} \log \rho(x_{a:b}) + \sum_{(a,b)\in\mathcal{P}} \sum_{(c,d)\in\mathcal{P}'(a,b)} \log \mu^{f(a)}(x_{c:d}|x_{a:c-1}) \\
&\leq \Gamma_d(\mathcal{P}') + \sum_{(a,b)\in\mathcal{P}'} g(b-a+1).
\end{aligned}$$

Since by definition $g$ is concave, Jensen's inequality implies $\sum_{(a,b)\in\mathcal{P}'} g(b-a+1) \leq |\mathcal{P}'| g(n/|\mathcal{P}'|)$. Furthermore, the conditions on $g$ also imply that $ag(b/a)$ is a nondecreasing function of $a > 0$ for any fixed $b > 0$, and so Lemma 2 implies

$$\sum_{(a,b)\in\mathcal{P}'} g(b-a+1) \leq |\mathcal{P}|\, g\left(\left\lceil \frac{n}{|\mathcal{P}|(\lceil \log n \rceil + 1)} \right\rceil\right)(\lceil \log n \rceil + 1),$$

which completes the main part of the proof. The term $\Gamma_d(\mathcal{P}')$ is proportional to the size of the partition tree needed to capture the locations where the data source changes. Since at least half of the nodes of a binary tree are leaves, and the number of leaf nodes is the same as the number of segments in a partition, for any $d$ we have $\Gamma_d(\mathcal{P}') \leq 2|\mathcal{P}'| \leq 2|\mathcal{P}|\,(\lceil \log n \rceil + 1)$. □

We also remark that Theorem 1 can be obtained by combining Lemma 1 in [9] with our Lemma 2 and the properties of $g$, as per the proof of Theorem 2 in [9]. However, to be self-contained, we decided to include the above short proof of the theorem.

**Removing the dependence on $n$ and $d$.** Our previous results required choosing a depth $d$ in advance such that $d = \lceil \log n \rceil$. This restriction can be lifted by using the modified coding distribution given by $\mathrm{PTW}(x_{1:n}) := \prod_{i=1}^{n} \mathrm{PTW}_{\lceil \log i \rceil}(x_i|x_{<i})$. The next result justifies this choice.

**Theorem 2.** *For all $n \in \mathbb{N}$, for all $x_{1:n} \in \mathcal{X}^n$, we have that*

$$-\log \mathrm{PTW}(x_{1:n}) \leq -\log \mathrm{PTW}_d(x_{1:n}) + \lceil \log n \rceil (\log 3 - 1),$$

*where $d = \lceil \log n \rceil$.*

Thus, the overhead due to not knowing $n$ in advance is $O(\log n)$. The proof of this result, given in Appendix A, is based on the fact that for any $t, k \in \mathbb{N}$ satisfying $1 \leq t \leq 2^k$, we have

$$\tfrac{2}{3}\mathrm{PTW}_{k+1}(x_{1:t}) \leq \mathrm{PTW}_k(x_{1:t}), \tag{3}$$

which implies that each time the depth of the tree is increased, the algorithm suffers at most an extra $\log(3/2)$ penalty.

Algorithm 1 can be straightforwardly modified to compute $\mathrm{PTW}(x_{1:n})$ using the same amount of resources as needed for $\mathrm{PTW}_d(x_{1:n})$. The main idea is to increase the size of the stack by one and copy over the relevant statistics whenever a power of two boundary is crossed. Alternatively, one could simply pick a sufficiently large value of $d$ in advance. This is also justified, as, based on Equation 3, the penalty for using an unnecessarily large $k > d$ can be bounded by

$$-\log \mathrm{PTW}_k(x_{1:n}) \leq -\log \mathrm{PTW}_d(x_{1:n}) + (k-d)\log \tfrac{3}{2}.$$

## 4 Applications

We now explore the performance of Partition Tree Weighting in a variety of settings.

**Binary, Memoryless, Piecewise Stationary Sources.** First we investigate using the well known KT estimator [13] as a base model for PTW. We begin with a brief overview of the KT estimator. Consider a sequence $x_{1:n} \in \{0,1\}^n$ generated by successive Bernoulli trials. If $a$ and $b$ denote the number of zeroes and ones in $x_{1:n}$ respectively, and $\theta \in [0,1]$ denotes the probability of observing a 1 on any given trial, then $\Pr(x_{1:n} \,|\, \theta) = \theta^b(1-\theta)^a$. One way to construct a distribution over $x_{1:n}$, in the case where $\theta$ is unknown, is to weight over the possible values of $\theta$. The KT-estimator uses the weighting $w(\theta) := \text{Beta}(\frac{1}{2},\frac{1}{2}) = \pi^{-1}\theta^{-1/2}(1-\theta)^{-1/2}$, which gives the coding distribution $\text{KT}(x_{1:n}) := \int_0^1 \theta^b(1-\theta)^a w(\theta)\, d\theta$ . This quantity can be efficiently computed online by maintaining the $a$ and $b$ counts incrementally and using the chain rule, that is, $\Pr(x_{n+1} = 1|x_{1:n}) = 1 - \Pr(x_{n+1} = 0|x_{1:n}) = (b + 1/2)/(n+1)$. Furthermore, the parameter redundancy can be bounded uniformly; restating a result from [22], one can show that for all $n \in \mathbb{N}$, for all $x_{1:n} \in \mathcal{X}^n$, for all $\theta \in [0,1]$,

$$\log \theta^b(1-\theta)^a - \log \text{KT}(x_{1:n}) \leq \tfrac{1}{2}\log(n) + 1. \tag{4}$$

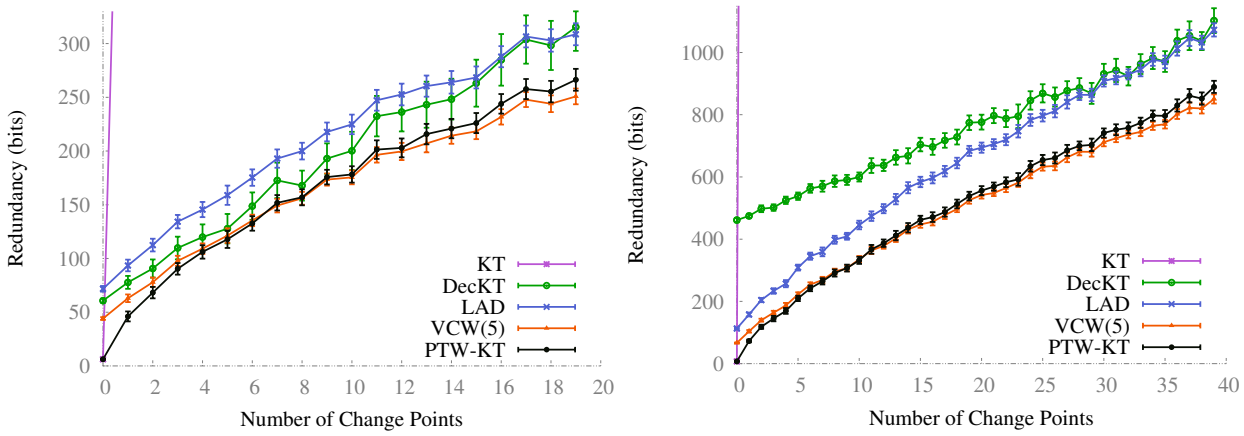
We now analyze the performance of the KT estimator when used in combination with PTW. Our next result follows immediately from Theorem 1 and Equation 4.

**Corollary 1.** *For all $n \in \mathbb{N}$, for all $x_{1:n} \in \{0,1\}^n$, if $\mu$ is a piecewise stationary source segmented according to any partition $\mathcal{P} \in \mathcal{T}_n$, with the data in segment $i$ being generated by i.i.d. Bernoulli$(\theta_i)$ trials with $\theta_i \in [0,1]$ for $1 \leq i \leq |\mathcal{P}|$, the redundancy of the* PTW-KT *algorithm, obtained by setting $d = \lceil \log n \rceil$ and using the* KT *estimator as a base model, is upper bounded by*

$$\Gamma_d(\mathcal{P}') + \frac{|\mathcal{P}|}{2}\log\left\lceil \frac{n}{|\mathcal{P}|(\lceil \log n \rceil + 1)} \right\rceil (\lceil \log n \rceil + 1) + |\mathcal{P}|(\lceil \log n \rceil + 1).$$

Corollary 1 shows that the redundancy behavior of PTW-KT is $O\left(|\mathcal{P}|(\log n)^2\right)$. Thus we expect this technique to perform well if the number of stationary segments is small relative to the length of the data. This bound also has the same asymptotic order as previous [20, 10, 9] low complexity techniques.

Next we present some experiments with PTW-KT on synthetic, piecewise stationary data. We compare against techniques from information theory and online learning, including (i) Live and Die Coding (LAD) [20], (ii) the variable complexity, exponential weighted averaging (VCW) method of [9], and (iii) the DEC-KT estimator [14, 17], a heuristic variant of the KT estimator that exponentially decays the $a$,$b$ counts to better handle non-stationary sources. Figure 3 illustrates the redundancy of each method as the number of change points increases. To mitigate the effects of any particular choice of change points or $\theta_i$, we report results averaged over 50 runs, with each run using a uniformly random set of change points locations and $\theta_i$. 95% confidence intervals are shown on the graphs. PTW performs noticeably better than all methods when the number of segments is small. When the number of segments gets high, both PTW and VCW outperform all other techniques, with VCW being slightly better once the number of change points is sufficiently large. The VCW$(g)$ technique interpolates between a weighting scheme very close to the linear method in [21] and LAD; however, the complexity of this algorithm when applied to the KT estimator is $O(gn \log n)$, so with $g = 5$, the runtime is already 5 times larger than PTW.




(a) $n = 8192$ over splits $\{0, 1, 2, \ldots, 20\}$ (b) $n = 65536$ over splits $\{0, 1, 2, \ldots, 40\}$

Figure 3: Average redundancy of various estimators on binary data for increasing number of change points.

| CTS + | bib | book1 | book2 | geo | news | obj1 | obj2 | paper1 | paper2 | paper3 | paper4 | paper5 | paper6 | pic | progc | progl | progp | trans |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DEC-KT | 1.78 | 2.18 | 1.88 | 4.30 | **2.31** | 3.67 | 2.31 | **2.25** | **2.20** | **2.45** | **2.75** | **2.87** | **2.33** | 0.78 | **2.28** | 1.60 | 1.62 | 1.36 |
| KT | 1.79 | **2.17** | 1.89 | 4.38 | 2.32 | 3.73 | 2.39 | **2.25** | **2.20** | **2.45** | 2.76 | 2.89 | 2.34 | 0.79 | 2.30 | 1.61 | 1.64 | 1.37 |
| LAD | 2.70 | 2.60 | 2.46 | 4.37 | 3.14 | 4.52 | 3.07 | 3.33 | 3.03 | 3.41 | 3.87 | 4.04 | 3.45 | 0.80 | 3.42 | 2.49 | 2.65 | 2.66 |
| VCW(5) | 2.32 | 2.41 | 2.19 | 4.27 | 2.80 | 4.13 | 2.72 | 2.90 | 2.69 | 3.02 | 3.43 | 3.56 | 3.00 | 0.78 | 2.95 | 2.11 | 2.22 | 2.16 |
| PTW-KT | **1.77** | **2.17** | **1.87** | **4.20** | **2.31** | **3.64** | **2.25** | **2.25** | **2.20** | **2.45** | **2.75** | 2.88 | **2.33** | **0.77** | 2.29 | **1.59** | **1.61** | **1.35** |

Table 1: Performance (average bits per byte) on the Calgary Corpus

Additionally, we evaluated the same set of techniques as a replacement to the KT estimator for the memoryless model used within Context Tree Switching (CTS) [17], a recently introduced universal data compression algorithm for binary, stationary Markov sources of bounded memory. Performance was measured on the well known Calgary Corpus [3]. Each result was generated using CTS with a context depth of 48 bits. The results (in average bits per byte) are shown in Table 1. Here we see that PTW-KTconsistently matches or outperforms the other methods. The largest relative improvements are seen on the non-text files, GEO, OBJ1, OBJ2 and PIC. While the performance of VCW could be improved by using a $g > 5$, it was already considerably slower than the other methods.

**Tracking.** PTW can also be used to derive an alternate algorithm for tracking [11] using the code-length loss. Consider a base model $\rho$ that is a convex combination of a finite set $\mathcal{M} := \{\nu_1, \nu_2, \ldots, \nu_{|\mathcal{M}|}\}$ of $k$-bounded memory models, that is,

$$\rho(x_{1:n} \,|\, x_{1-k:0}) := \sum_{\nu_i \in \mathcal{M}} w_{\nu_i} \nu_i(x_{1:n} \,|\, x_{1-k:0}), \tag{5}$$

where $x_{1-k:0} \in \mathcal{X}^k$ denotes the initial (possibly empty) context, each $\nu_i$ is a $k$-bounded memory probabilistic data generating source (that is, $\nu_i(x_t|x_{<t}) = \nu_i(x_t|x_{t-k:t-1})$ for any $t$), $w_\nu \in \mathbb{R}$ and $w_\nu > 0$ for all $\nu \in \mathcal{M}$, and $\sum_{\nu \in \mathcal{M}} w_\nu = 1$. We now show that applying PTW to $\rho$ gives rise to a model that will perform well with respect to an interesting subset of the class of switching models. A switching model is composed of two parts, a set of models $\mathcal{M}$ and an index set. An index set $i_{1:n}$ with respect to $\mathcal{M}$ is an element of $\{1, 2, \ldots, |\mathcal{M}|\}^n$. Furthermore, an index set $i_{1:n}$ can be

naturally mapped to a temporal partition in $\mathcal{T}_n$ by processing the index set sequentially, adding a new segment whenever $i_t \neq i_{t+1}$ for $1 \leq t < n$. For example, if $|\mathcal{M}| \geq 2$, the string $1111122221$ maps to the temporal partition $\{(1,5),(6,9),(10,10)\}$. The partition induced by this mapping will be denoted by $\mathcal{S}(i_{1:n})$. A switching model can then be defined as

$$\xi_{i_{1:n}}(x_{1:n}) := \prod_{(a,b)\in\mathcal{S}(i_{1:n})} \nu_{i_a}(x_{a:b} \,|\, x_{a-k:a-1}),$$

where we have adopted the convention that the previous symbols at each segment boundary define the initializing context for the next bounded memory source.[2] The set of all possible switching models for a sequence of length $n$ with respect to the model class $\mathcal{M}$ will be denoted by $\mathcal{I}_n(\mathcal{M})$. If we now let $\tau(x_{1:n})$ denote $\text{PTW}_{\lceil \log n \rceil}(x_{1:n})$ using a base model as defined by Equation 5, we can state the following upper bound on the redundancy of $\tau$ with respect to an arbitrary switching model.

**Corollary 2.** *For all $n \in \mathbb{N}$, for any $x_{1:n} \in \mathcal{X}^n$ and for any switching model $\xi_{i_{1:n}} \in \mathcal{I}_n(\mathcal{M})$, we have*

$$-\log \tau(x_{1:n}) + \log \xi_{i_{1:n}}(x_{1:n}) \leq (2+\kappa)\,|\mathcal{S}(i_{1:n})|\,(\lceil \log n \rceil + 1), \tag{6}$$

*where $\kappa := \max_{\nu\in\mathcal{M}} -\log(w_\nu)$.*

*Proof.* Using $-\log \rho(x_{1:t} \,|\, x_{1-k:0}) = -\log\left\{\sum_{\nu_i\in\mathcal{M}} w_{\nu_i}\nu_i(x_{1:t} \,|\, x_{1-k:0})\right\} \leq -\log w_{\nu^*} - \log \nu^*(x_{1:t} \,|\, x_{1-k:0})$ for any $\nu^* \in \mathcal{M}$ and $t \in \mathbb{N}$, we see that the redundancy of $\rho$ with respect to any single model in $\mathcal{M}$ is bounded by $\kappa := \max_{\nu\in\mathcal{M}} -\log(w_\nu)$. Combining this with Theorem 1 completes the proof. □

Inspecting Corollary 2, we see that there is a linear dependence on the number of change points and a logarithmic dependence on the sequence length. Thus we can expect our tracking technique to perform well provided the data generating source can be well modeled by some switching model that changes infrequently. The main difference between our method and [11] is that our prior depends on additional structure within the index sequence. While both methods have a strong prior bias towards favoring a smaller number of change points, the PTW prior arguably does a better job of ensuring that the change points are not clustered too tightly together. This benefit does however require logarithmically more time and space.

## 5 Extensions and Future Work

Along the lines of [9], our method could be extended to a more general class of loss functions within an online learning framework. We also remark that a technique similar to Context Tree Maximizing [18] can be applied to PTW to extract the best, in terms of Minimum Description Length [8], binary temporal partition for a given sequence of data. Unfortunately we could not find a way to avoid building a full context tree for this case, which means that $O(n \log n)$ memory would be required instead of the $O(\log n)$ required by Algorithm 1. Another interesting follow up would be to generalize Theorem 3 in [17] to the piecewise stationary setting. Combining such a result with Corollary 1 would allow us to derive a redundancy bound for the algorithm that uses

[2] This is a choice of convenience. One could always relax this assumption and naively encode the first $k$ symbols of any segment using a uniform probability model, incurring a startup cost of $k \log |\mathcal{X}|$ bits before applying the relevant bounded memory model. This would increase the upper bound in Equation 6 by $|\mathcal{S}(i_{1:n})|\,(\lceil \log n \rceil + 1)k \log |\mathcal{X}|$.

PTW-KT for the memoryless model within CTS; this bound would hold with respect to any binary, piecewise stationary Markov source of bounded memory.

# 6 Conclusion

This paper has introduced Partition Tree Weighting, an efficient meta-algorithm that automatically generalizes existing coding distributions to their piecewise stationary extensions. Our main contribution is to introduce a prior, closely related to the Context Tree Weighting method, to efficiently weight over a large subset of possible temporal partitions. The order of the redundancy and the complexity of our algorithm matches those of the best competitors available in the literature, with the new algorithm exhibiting a superior complexity-performance trade-off in our experiments.

**Acknowledgments.** The authors would like to thank Marcus Hutter for some helpful comments. This research was supported by NSERC and Alberta Innovates Technology Futures.

# References

[1] S. Adak. Time-dependent spectral analysis of nonstationary time series. *Journal of the American Statistical Association*, pages 1488–1501, 1998.

[2] D. Angelosante and G.B. Giannakis. Sparse graphical modeling of piecewise-stationary time series. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 1960–1963. IEEE, 2011.

[3] Ross Arnold and Tim Bell. A corpus for the evaluation of lossless compression algorithms, 1997.

[4] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA, 2006. ISBN 0521841089.

[5] R.A. Davis, T.C.M. Lee, and G.A. Rodriguez-Yam. Structural break estimation for nonstationary time series models. *Journal of the American Statistical Association*, 101(473):223–239, 2006.

[6] Steven de Rooij and Tim van Erven. Learning the switching rate by discretising Bernoulli sources online. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, volume 5 of *JMLR Workshop and Conference Proceedings*, pages 432–439, Clearwater Beach, Florida USA, April 2009.

[7] Paul Fearnhead. Exact and efficient bayesian inference for multiple changepoint problems. *Statistics and Computing*, 16(2):203–213, 2006.

[8] Peter D. Grünwald. *The Minimum Description Length Principle (Adaptive Computation and Machine Learning)*. The MIT Press, 2007. ISBN 0262072815.

[9] A. György, T. Linder, and G. Lugosi. Efficient tracking of large classes of experts. *IEEE Transactions on Information Theory*, 58(11):6709–6725, 2011.

[10] E. Hazan and C. Seshadhri. Efficient learning algorithms for changing environments. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 393–400. ACM, 2009.

[11] Mark Herbster and Manfred K. Warmuth. Tracking the best expert. *Machine Learning*, 32:151–178, August 1998.

[12] Wouter Koolen and Steven de Rooij. Combining expert advice efficiently. In *Proceedings of the 21st Annual Conference on Learning Theory, COLT 2008*, pages 275–286, Helsinki, Finland, July 2008.

[13] R. Krichevsky and V. Trofimov. The performance of universal encoding. *Information Theory, IEEE Transactions on*, 27(2):199–207, 1981.

[14] Alexander O'Neill, Marcus Hutter, Wen Shao, and Peter Sunehag. Adaptive context tree weighting. In *DCC*, pages 317–326, 2012.

[15] R. Prescott Adams and D. J. C. MacKay. Bayesian Online Changepoint Detection. *ArXiv e-prints*, 2007.

[16] Gil I. Shamir and Neri Merhav. Low Complexity Sequential Lossless Coding for Piecewise Stationary Memoryless Sources. *IEEE Transactions on Information Theory*, 45:1498–1519, 1999.

[17] Joel Veness, Kee Siong Ng, Marcus Hutter, and Michael H. Bowling. Context tree switching. *Data Compression Conference*, 2012.

[18] Paul A. J. Volf and Frans M. J. Willems. Context maximizing: Finding MDL decision trees. In *In Symposium on Information Theory in the Benelux, Vol.15*, pages 192–200, 1994.

[19] V. Vovk. Derandomizing stochastic prediction strategies. *Machine Learning*, 35(3):247–282, Jun. 1999.

[20] F. Willems and M. Krom. Live-and-die coding for binary piecewise i.i.d. sources. In *Information Theory. 1997. Proceedings., 1997 IEEE International Symposium on*, page 68, jun-4 jul 1997. doi: 10.1109/ISIT.1997.612983.

[21] Frans M. J. Willems. Coding for a binary independent piecewise-identically-distributed source. *IEEE Transactions on Information Theory*, 42:2210–2217, 1996.

[22] Frans M.J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. The Context Tree Weighting Method: Basic Properties. *IEEE Transactions on Information Theory*, 41:653–664, 1995.

[23] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, Washington, DC, USA, 2003.

## Appendix A. Supplementary Proofs

**Lemma 1.** *For any depth $d \in \mathbb{N}$, given a sequence of data $x_{1:n} \in \mathcal{X}^n$ satisfying $n \leq 2^d$,*

$$\text{PTW}_d(x_{1:n}) = \frac{1}{2}\rho(x_{1:n}) + \frac{1}{2}\text{PTW}_{d-1}\left(x_{1:k}\right)\text{PTW}_{d-1}\left(x_{k+1:n}\right), \tag{2}$$

*where $k = 2^{d-1}$.*

*Proof.* This is a straightforward adaptation of Lemma 2 from [22]. We use induction on $d$. First note that $\text{PTW}_0(x_{1:n}) = \rho(x_{1:n})$ by definition, so the base case of $d = 1$ holds trivially. Now assume that Equation 2 holds for some depth $d-1$, and observe that

$$\begin{aligned}
\text{PTW}_d(x_{1:n}) &= \sum_{\mathcal{P}\in\mathcal{C}_d(1)} 2^{-\Gamma_d(\mathcal{P})} \prod_{(i,j)\in\mathcal{P}} \rho(x_{i:j}) \\
&= \tfrac{1}{2}\rho(x_{1:n}) + \sum_{\mathcal{P}\in\mathcal{C}_d(1)\setminus\{\{(1,2^d)\}\}} 2^{-\Gamma_d(\mathcal{P})} \prod_{(i,j)\in\mathcal{P}} \rho(x_{i:j}) \\
&= \tfrac{1}{2}\rho(x_{1:n}) + \tfrac{1}{2} \sum_{\substack{\mathcal{P}_1\in\mathcal{C}_{d-1}(1)\\ \mathcal{P}_2\in\mathcal{C}_{d-1}(k+1)}} 2^{-\Gamma_{d-1}(\mathcal{P}_1)-\Gamma_{d-1}(\mathcal{P}_2)} \prod_{(i,j)\in\mathcal{P}_1} \rho(x_{i:j}) \prod_{(r,s)\in\mathcal{P}_2} \rho(x_{r:s}) \\
&= \tfrac{1}{2}\rho(x_{1:n}) + \tfrac{1}{2} \left( \sum_{\mathcal{P}_1\in\mathcal{C}_{d-1}(1)} 2^{-\Gamma_{d-1}(\mathcal{P}_1)} \prod_{(i,j)\in\mathcal{P}_1} \rho(x_{i:j}) \right) \left( \sum_{\mathcal{P}_2\in\mathcal{C}_{d-1}(k+1)} 2^{-\Gamma_{d-1}(\mathcal{P}_2)} \prod_{(r,s)\in\mathcal{P}_2} \rho(x_{r:s}) \right) \\
&= \frac{1}{2}\rho(x_{1:n}) + \frac{1}{2}\text{PTW}_{d-1}\left(x_{1:k}\right)\text{PTW}_{d-1}\left(x_{k+1:n}\right)
\end{aligned}$$

where $k := 2^{d-1}$. The third step uses the property $\Gamma_d(\mathcal{P}_1 \cup \mathcal{P}_2) = \Gamma_{d-1}(\mathcal{P}_1) + \Gamma_{d-1}(\mathcal{P}_2) + 1$, which holds when $\mathcal{P}_1 \in \mathcal{C}_{d-1}(1)$ and $\mathcal{P}_2 \in \mathcal{C}_{d-1}(k+1)$. The final step applies the inductive hypothesis. □

**Theorem 2.** *For all $n \in \mathbb{N}$, for all $x_{1:n} \in \mathcal{X}^n$, we have that*

$$-\log \text{PTW}(x_{1:n}) \leq -\log \text{PTW}_d(x_{1:n}) + \lceil \log n \rceil (\log 3 - 1),$$

*where $d = \lceil \log n \rceil$.*

*Proof.* We begin by showing that for any $t \leq 2^k$ we have

$$\tfrac{2}{3}\text{PTW}_{k+1}(x_{1:t}) \leq \text{PTW}_k(x_{1:t}). \tag{7}$$

Using Lemma 1,

$$\text{PTW}_{k+1}(x_{1:t}) = \frac{1}{2}\rho(x_{1:t}) + \frac{1}{2}\text{PTW}_k\left(x_{1:2^k}\right)\text{PTW}_k\left(x_{2^k+1:2^{k+1}}\right) = \frac{1}{2}\rho(x_{1:t}) + \frac{1}{2}\text{PTW}_k\left(x_{1:t}\right) \tag{8}$$

holds for $t \leq 2^k$. On the other hand,

$$\text{PTW}_k(x_{1:t}) = \sum_{\mathcal{P}\in\mathcal{C}_k} 2^{-\Gamma_k(\mathcal{P})} \prod_{(a,b)\in\mathcal{P}} \rho(x_{a:b}) = \frac{1}{2}\rho(x_{1:t}) + \sum_{\mathcal{P}\in\mathcal{C}_k\setminus\{\{(1,2^k)\}\}} 2^{-\Gamma_k(\mathcal{P})} \prod_{(a,b)\in\mathcal{P}} \rho(x_{a:b}) \geq \tfrac{1}{2}\rho(x_{1:t}),$$

which, together with Equation 8 proves Equation 7. The latter allows us to derive a lower bound on $\text{PTW}(x_{1:n})$, by noting that

$$\begin{aligned}
\text{PTW}(x_{1:n}) &= \prod_{i=1}^{n} \text{PTW}_{\lceil \log i \rceil}(x_i|x_{<i}) \\
&= \text{PTW}_0(x_1) \left( \prod_{a=1}^{d-1} \text{PTW}_a(x_{2^{a-1}+1:2^a}|x_{1:2^{a-1}}) \right) \text{PTW}_d(x_{2^{d-1}+1:n}|x_{1:2^{d-1}}) \\
&= \text{PTW}_0(x_1) \left( \prod_{a=1}^{d-1} \frac{\text{PTW}_a(x_{1:2^a})}{\text{PTW}_a(x_{1:2^{a-1}})} \right) \frac{\text{PTW}_d(x_{1:n})}{\text{PTW}_d(x_{1:2^{d-1}})} \\
&= \text{PTW}_d(x_{1:n}) \prod_{a=1}^{d} \frac{\text{PTW}_{a-1}(x_{1:2^{a-1}})}{\text{PTW}_a(x_{1:2^{a-1}})} \\
&\geq \left(\tfrac{2}{3}\right)^d \text{PTW}_d(x_{1:n}).
\end{aligned}$$

Hence, taking the negative logarithm of both sides we obtain

$$-\log \text{PTW}(x_{1:n}) \leq -\log \text{PTW}_d(x_{1:n}) + d \log(\tfrac{3}{2}) = -\log \text{PTW}_d(x_{1:n}) + \lceil \log n \rceil (\log 3 - 1).$$

□